Operations on (ordered) interval sets

E.O. de Brock

SOM theme A
The human and technical side of production:
the management of interdependencies

Abstract

Intervals play an important role in various kinds of database-applications in practice, for example in historical, spatial, and temporal databases. As a consequence, there is a practical need for a clear and proper treatment of various useful operations on intervals and interval sets in a database context. However, the semantics of some important operations on interval sets are not always treated or not treated very clearly in the literature; e.g., often they are defined in an algorithmic rather than a declarative manner. Moreover, implementation proposals are often not as straightforward as they could be.

This paper presents a declarative treatment of various operations on interval sets, also introducing some new notions (such as *ordered* interval sets, their *visible* points, and their *surface*). Then the paper formally "links" such (mathematical) intervals to their database representations. Finally the paper provides straightforward translations from these formal database representations to standard SQL, without the need for SQL extensions.

Keywords: Interval set, minimal and maximal interval, ordered interval set, visibility, surface, interval representation, realization in SQL

(also downloadable) in electronic version: http://som.rug.nl/

Introduction

Intervals play an important role in various kinds of database-applications. Well-known classes of examples are spatial, temporal, spatio-temporal, and historical databases; many papers and models regarding these types of interval databases have appeared. See for instance the bibliographies [AS93] and [WJ98], the books [TC93] and [EJ98] with selected papers, and the surveys in [MS91] and [LM95]. As already emphasized in [LM95], such domain specific application classes can in fact be generalized to arbitrary intervals. There is a practical need for a clear treatment of various useful operations on intervals and interval sets in a database context. Examples of such operations are the union and the intersection of the point sets of two interval sets. Other examples relate to what we call *ordered* interval sets, where intervals are ordered (e.g., in time or in place) and "newer" intervals "overrule" the "older" intervals (partly or completely). Given an ordered interval set, we want to define its set of *visible* points, its set of *visible* intervals, and its *surface*, for instance. These notions are important in practical applications of ordered interval sets.

In the literature, the semantics of operations on interval sets (such as "union" and "intersection") are not always treated very well (as also noted in [LP94] and [To00]). Often the semantics of such operations are defined in an imperative, algorithmic manner in stead of a declarative manner (see [Ar86] or the fold operation in [LP94] for instance). In other cases, the definitions are not straightforward (such as the definition of a canonical relation in [LP95]). Moreover, existing proposals are often not as straightforward as possible and may require extensions to the data model ([LJ88], [TC93]) and/or to SQL itself ([NA93], [Sa90], [To00]). In particular, TSQL2 is a temporal extension of SQL-92 ([Sn95], [SB98]).

In order to address these problems properly, it is useful to make a clear distinction between abstract (mathematical) intervals and their (database) representations. This paper therefore starts with a formal mathematical treatment of intervals and several related notions (including the new notions related to *ordered* interval sets). Our collection (and order) of definitions is carefully chosen here, in order to be as (relatively) simple as possible. We will concentrate on left-closed, right-open intervals (as usual in the literature). Then we introduce the general notion of a *representation*, in order to "link" intervals to (relational) databases. Finally we show how our notions and constructs can be translated into standard SQL. So, schematically we have (where "→" means "are represented by"):

real world objects → mathematical intervals → database representations → SQL realizations

All in all, it turns out that what need to be extended is not so much the (relational) database model itself, but our comprehension and use of that model!

The paper is organized as follows. Section 1 contains a formal mathematical treatment of intervals and related notions (over any lineair domain). Section 2 introduces the general notion of representations, in particular representations of (ordered) interval sets. We subsequently zoom in on *tables* as representations of (ordered) interval sets. We explain how the interval-related notions from Section 1 are simulated by their table representatives. Section 3 explains how the notions and constructs from Section 2 can be translated into standard SQL. Section 4 compares our approach with other approaches in the literature. The paper ends with some conclusions.

1. Mathematical model

This section presents the necessary mathematical background. Our formalization of the basic notions starts with lineair domains and intervals in Section 1.1, continues with interval sets in Section 1.2, and ends with ordered interval sets in Section 1.3. Our collection and order of definitions is carefully chosen, in order to be as (relatively) simple as possible.

1.1. Lineair domains and intervals

A *lineair domain* is a pair (D;<) consisting of a set D and a lineair (or total) order < on D, i.e., < is irreflexive, transitive, and connected. Examples of lineair domains are integer sets and real number sets (with cardinal order), date sets and other time sets (with chronological order), and string sets (with lexicographical order). We do not need to require that a lineair domain is finite or even discrete. Also our theory does not require any specific granularity constraints for the domain in advance. Therefore, our theory can support any granularity.

As usual, $a \leq b$ stands for "$a < b$ or $a = b$". We note that the choice between "<" and "≤" in our definitions will be very important.

Let $V \subseteq D$. We write "$x \lhd y$ in V" iff $x < y$ and there does not exist a member of V strictly between x and y. We then say that x *preceeds* y *directly in* V (or y *succeeds* x *directly in* V). Formally:

$x \lhd y$ in V iff $x < y$ and $\neg \exists z \in V: (x < z \text{ and } z < y)$.

For $b \in D$ and $e \in D$ with $b < e$, the *interval* [b..e) over (D;<) is defined as follows:
$\underline{[b..e)} = \{ d \in D \mid b \leq d \text{ and } d < e \}$.

So, [b..e) is a subset of D. Note that $b \in [b..e)$ and $e \notin [b..e)$. Also note that $[b..e) = [b'..e')$ if and only if $b = b'$ and $e = e'$. We note that [b..e) is a left-closed, right-open interval. We call b *the begin point* and e *the end point* of the interval [b..e); b and e are also called the *boundary points* of [b..e).

Since intervals are sets, set-related notions also apply to intervals. We say that

(a) [b..e) and [b′..e′) are *disjoint* iff $[b..e) \cap [b'..e') = \emptyset$, which reduces to $e \leq b'$ or $e' \leq b$.

(b) [b..e) and [b′..e′) *intersect* iff they are not disjoint, which reduces to $b' < e$ and $b < e'$.

(c) [b..e) *includes* or *covers* [b′..e′) iff $[b'..e') \subseteq [b..e)$, which reduces to $b \leq b'$ and $e' \leq e$.

(d) [b..e) and [b′..e′) are *adjacent* iff $b' = e$ or $b = e'$.
So, adjacent intervals are disjoint.

(e) [b..e) and [b′..e′) are *separate* iff they are disjoint and not adjacent (so $e < b'$ or $e' < b$).

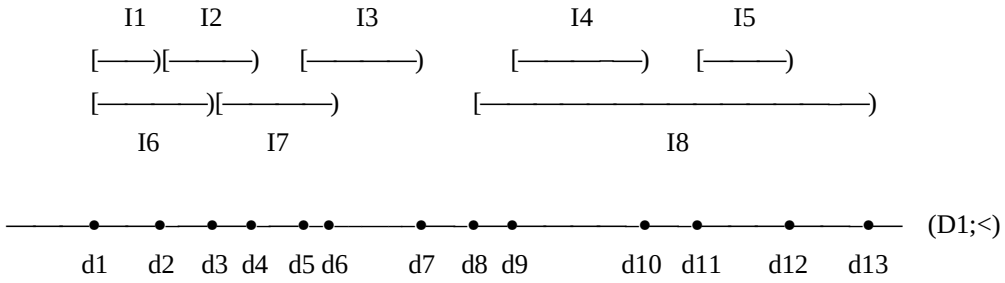

Figure 1: A set of intervals over a lineair domain (D1;<)

As an illustration, Figure 1 contains 8 intervals over some lineair domain (D1;<). For the sake of clarity, we did not draw all intervals on the same line. The intervals I1 and I2 are adjacent (and hence disjoint), I2 and I3 are separate, I2 and I6 intersect (as well as I1 and I6), and I6 includes I1.

1.2. Interval sets

Let S be a set of intervals over (D;<). We call S *intersection-free* iff its intervals are mutually disjoint:
S is *intersection-free* iff $\forall I \in S: \forall I' \in S$: if $I \neq I'$ then I and I′ are disjoint.
In line with the intention in [LP95], we call S *canonical* iff its intervals are mutually separate:
S is *canonical* iff $\forall I \in S: \forall I' \in S$: if $I \neq I'$ then I and I′ are separate.
So, if S is canonical then S is also intersection-free. Also, if S is canonical or intersection-free then so is any subset of S.
Let S1 be the set of 8 intervals in Figure 1. The interval set S1 is not intersection-free (and hence not canonical), its subset {I1, I2, I3, I4, I5} is intersection-free but not canonical, and the subset {I2, I3, I4} is canonical (and hence intersection-free).

If X is any set of sets then $\bar{\cup}$ X, *the point set* (or *generalized union*) *of* X, is the union of all members of X: $\bar{\cup} X = \{ y \mid \exists Y \in X: y \in Y \}$. In particular, this definition also applies if X is a set of intervals, or even multi-dimensional intervals (see [LP94] for instance). Returning to the interval set S1, we see that $\bar{\cup}$ S1 = [d1..d7) $\cup$ [d8..d13), which is a subset of D1.
Let X and X′ be sets of sets. We call X *point-wise equivalent to* X′ iff their point sets are the same, i.e., iff $\bar{\cup} X = \bar{\cup} X'$. Let S2 be the interval set {[d1..d7), [d8..d13)}. We note that S2 is canonical and point-wise equivalent to S1.

We define Begset(S), the set of all *begin points* in S, Endset(S), the set of all *end points* in S, and Bndset(S), the set of all *boundary points* (i.e., begin and end points) in S, as follows:
Begset(S) = { b | [b..e) $\in$ S }
Endset(S) = { e | [b..e) $\in$ S }
Bndset(S) = Begset(S) $\cup$ Endset(S)

For the set S1 of intervals in Figure 1, for example, Begset(S1) has 7 elements (because two intervals have the begin point d1 in common), Endset(S1) has 8 elements, and Bndset(S1) - see Figure 2 - has 13 elements (because Begset(S1) and Endset(S1) have the two elements d2 and d3 in common).

A *minimal interval within* S is determined by a pair of successive boundary points of S, covered by some interval in S. We therefore define Minintset(S), the set of all minimal intervals within S, as:

Minintset(S) = { [x..y) | x ∈ Bndset(S) and y ∈ Bndset(S) and x ◁ y in Bndset(S) and
∃I ∈ S: I includes [x..y) }.

Figure 2 shows that S1 has 11 minimal intervals (J1 – J11). The requirement in the definition of Minintset(S) that some I in S includes [x..y), is also necessary; otherwise, in Figure 2 for instance, we would also have an interval between J6 and J7.

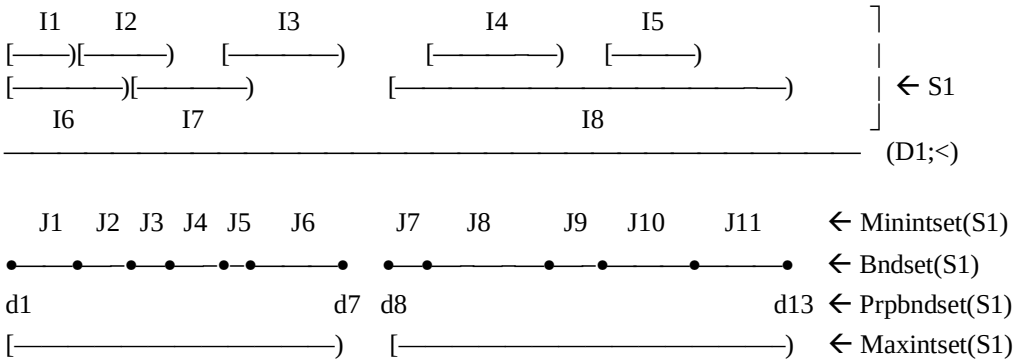

Figure 2: The interval set S1 and its minimal and maximal intervals

The following proposition summarizes the most important properties of the operation Minintset:

Proposition 1: If S is a finite set of intervals over (D;<) then:

(a) Minintset(S) is a finite set of intervals over (D;<).
(b) Minintset(S) is intersection-free.
(c) Minintset(S) is point-wise equivalent to S.
(d) Each interval in Minintset(S) is a subset of all its intersecting intervals in S.
(e) Each interval in Minintset(S) is the intersection of all its intersecting intervals in S.
(f) If S is intersection-free then Minintset(S) = S.
(g) Minintset(Minintset(S)) = Minintset(S).

We say that b is a *proper begin point* in S iff b is a begin point in S and every interval with a begin point smaller than b also has its end point smaller than b. Similarly, we say that e is a *proper end point* in S iff e is an end point in S and every interval with an end point larger than e also has its begin point larger than e. We say that x is a *proper boundary point* in S iff x is a proper begin point or a proper end point in S. So we define Prpbegset(S), the set of all *proper* begin points in S, Prpendset(S), the set of all *proper* end points in S, and Prpbndset(S), the set of all *proper* boundary points in S, as follows:

Prpbegset(S) = { b | b ∈ Begset(S) and ∀[b′..e′) ∈ S: (if b′ < b then e′ < b) }
Prpendset(S) = { e | e ∈ Endset(S) and ∀[b′..e′) ∈ S: (if e < e′ then e < b′) }
Prpbndset(S) = Prpbegset(S) ∪ Prpendset(S)

For the interval set S1 in Figure 2, Prpbegset(S1) = {d1, d8}, Prpendset(S1) = {d7, d13}, and hence Prpbndset(S1) = {d1, d7, d8, d13}.

A *maximal interval within* S is determined by a proper begin point b and a proper end point e in S which succeeds b directly in the set of all proper end points. We therefore define Maxintset(S), the set of all maximal intervals within S, as follows:

Maxintset(S) = { [b..e) | b ∈ Prpbegset(S) and e ∈ Prpendset(S) and b ◁ e in Prpendset(S) }

We saw that the interval set S1 has 2 proper begin points, 2 proper end points, and 4 proper boundary points. Figure 2 shows that S1 has 2 maximal intervals, namely [d1..d7) and [d8..d13). Hence, Maxintset(S1) equals the interval set S2 introduced before.
The following proposition summarizes the most important properties of the operation Maxintset:

Proposition 2: If S is a finite set of intervals over (D;<) then:
(a) Maxintset(S) is a finite set of intervals over (D;<).
(b) Maxintset(S) is canonical.
(c) Maxintset(S) is point-wise equivalent to S.
(d) Each interval in Maxintset(S) is a superset of all its intersecting intervals in S.
(e) Each interval in Maxintset(S) is the union of all its intersecting intervals in S.

(f) If S is canonical then Maxintset(S) = S.
(g) Maxintset(Maxintset(S)) = Maxintset(S).
(h) If S′ is a finite set of intervals over (D;<) and S′ is point-wise equivalent to S then Maxintset(S′) = Maxintset(S).
(i) Maxintset(Minintset(S)) = Maxintset(S).
(j) Minintset(Maxintset(S)) = Maxintset(S).

Where other authors define the "union" as a binary operation (e.g., P-union[V] (R, S) in [LPS95], where V denotes the set of interval attributes of R and S), we defined it as a unary operation, Maxintset(S). In case we have two interval sets, say S1 and S2, we can simply take the set of all maximal intervals within S1 ∪ S2, i.e., Maxintset(S1 ∪ S2); and similarly for more than two interval sets. More generally, if we would have a *set* of interval sets, say a set W, then we can take the set of all maximal intervals within the generalized union of W: Maxintset(Ū W). So, the unary Union operation is sufficient, and even more general than the usual binary one.
A similar remark holds for Minintset with respect to "intersection".

1.3. Ordered interval sets

Sometimes intervals are mutually ordered, e.g. in time. As a concrete practical example, the intervals may represent road layer segments that are ordered by lay-out date (as in [Br92]). As a more general class of examples, the intervals may represent time periods (in the past, present, or future) over which we think we know something (e.g., the presumed historical addresses of a suspect, or a planning), ordered by the time we first thought that. This is related to the notion of *simultaneous values* in [EG98], for instance. After Figure 3 below we will work out some of these examples, which we encountered in practice, and subsequently use them as running examples in the paper.
We can model an *ordered interval set* as a set X of pairs (I;t) where I is an interval from a lineair domain (D;<) and t is an element of a lineair domain (T;≺). We denote the set of intervals proper by:
Intset(X) = { I | (I;t) ∈ X }.

As an example, Figure 3 contains an ordered set X1 of 8 intervals, associated with the elements t1, t2, and t3 from T1; here t1 ≺ t2 and t2 ≺ t3. In this case we call (I1;t3) *more recent than* (I6;t1) and we call (I1;t3) and (I5;t3) *the most recent* intervals. We note that Intset(X1) = S1 (from Figure 2).

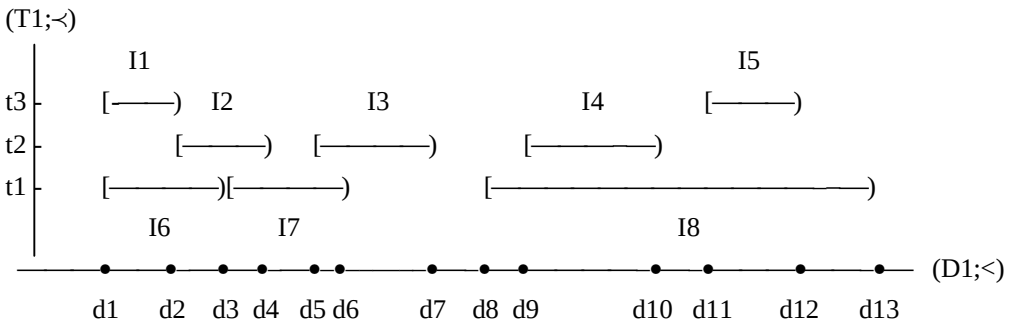

Figure 3: An ordered set of intervals

In case of our road layer segments example mentioned above, Figure 3 might be interpreted as follows:
At time t1 (e.g., t1 = 1996), a road was constructed from point d1 to point d3 and, using different surface material, also from d3 to d6 and from d8 to d13.
At time t2 (e.g., t2 = 1998), some parts of the road already needed a renewal of their surface material (maybe due to a severe winter), and the road was also extended to point d7.
At time t3 (e.g., t3 = 2000), the sections from d1 to d2 and from d11 to d12 were renewed.

In case of our presumed historical addresses example mentioned above (which we encountered in a social security environment of a large city), Figure 3 might be interpreted as follows:
At time t1 we received the following information about a certain citizen: (a) he resided at a given address from day d1 to day d3, (b) then he moved to another given address where he lived until day d6, and (c) he resided from (at least) day d8 to day d13 at some other given address.

At time t2 we heard about our citizen (a) that from day d2 to d4 he actually lived at another address than we thought, (b) what his actual address from day d5 to d7 was, and (c) that from day d9 to d10 he temporarily resided at another address than the presumed one.
At time t3 it turned out that the presumed addresses from day d1 to d2 and from day d11 to d12 were wrong and had to be overruled by other given addresses.

In our examples above we can address the question what the current (cumulative) road surface or our current (cumulative) perception of the presumed address history of our citizen is (after all those cumulative corrections). Although this question is intuitively clear, the general solution is not easily formalised. We will present our general formal solution below.

Informally, Visbegset(X), the set of all *begin points visible from above* (or shortly *visible begin points*) within an ordered interval set X consists of those begin points that are not "superseeded by" a more recent interval. We define *visible end points* similarly. A *visible boundary point* is a visible begin or end point. In the definitions below we use the following (closed interval) notation:
$[b..e] = \{ d \in D \mid b \leq d \text{ and } d \leq e \}$.

$\text{Visbegset}(X) = \{ b \mid ([b..e);t) \in X \text{ and } \neg \exists ([b'..e');t') \in X: (b \in [b'..e'] \text{ and } t \prec t') \}$
$\text{Visendset}(X) = \{ e \mid ([b..e);t) \in X \text{ and } \neg \exists ([b'..e');t') \in X: (e \in [b'..e'] \text{ and } t \prec t') \}$
$\text{Visbndset}(X) = \text{Visbegset}(X) \cup \text{Visendset}(X)$

A *visible interval* within X is "assembled" from two successive visible boundary points x and y and the second (or "time") component of the most recent ordered interval that covers [x..y). We therefore define Surface(X), the set of all visible intervals within X, as follows:

$$\underline{\text{Surface(X)}} = \{ ([x..y);t) \mid x \in \text{Visbndset}(X) \text{ and } y \in \text{Visbndset}(X) \text{ and } (I;t) \in X \text{ and } I \text{ covers } [x..y) \text{ and } x \lhd y \text{ in Visbndset}(X) \text{ and } \neg \exists (I';t') \in X: I' \text{ covers } [x..y) \text{ and } t \prec t' \}$$

Note that for a given pair of successive visible boundary points x and y in the definition of Surface(X), there is only one corresponding (most recent) t. In general, however, the underlying I that is mentioned in that definition need not be uniquely determined. On the other hand, practical applications often have the business rule that two different but equally recent

intervals within X have to be disjoint, or, in other words, that equally recent intersecting intervals within X must actually be the same intervals. Formally:

$$\forall(I;t) \in X: \forall(I';t') \in X: \text{if } t = t' \text{ and } I \cap I' \neq \emptyset \text{ then } I = I'. \qquad \text{(BR1)}$$

In that case, the underlying I is therefore uniquely determined for such a given pair of successive visible boundary points. Note that our example X1 in Figure 3 happens to satisfy this constraint.

Figure 4 shows that the interval set X1 from Figure 3 has 5 visible begin points (d1, d5, d8, d9, and d11), 6 visible end points, and 11 visible boundary points. (Note that the boundary points d3 and d6 are not visible.) Surface(X1) consists of 9 visible intervals (V1 – V9). For instance, V3 = ([d4..d5); t1).

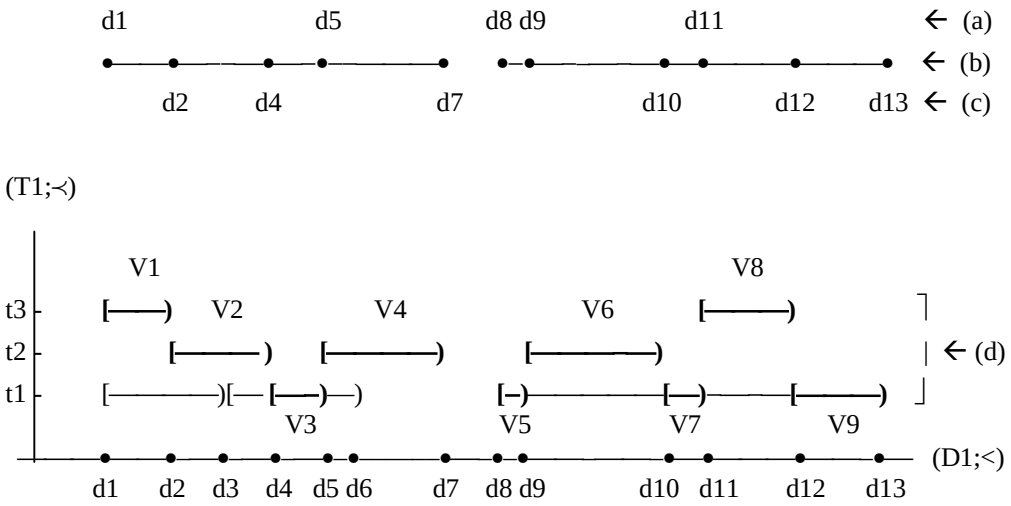

Figure 4: The visible points and visible intervals of an ordered set of intervals
(a) Visbegset(X1), (b) Visbndset(X1), (c) Visendset(X1), (d) Surface(X1)

The following proposition summarizes the most important properties of the Surface operation:

Proposition 3: If X is a finite, ordered set of intervals then:
(a) Surface(X) is a finite, ordered set of intervals.
(b) Intset(Surface(X)) is intersection-free.
(c) Intset(Surface(X)) is point-wise equivalent to Intset(X).
(d) Surface(Surface(X)) = Surface(X).

Since we keep track of the road surface history and the address history in our running examples, we should in principle be able to reconstruct the road surface and our knowledge about the address history of our citizen at any earlier moment in time, by using our formal Surface-operation. We illustrate this use by treating the query that asks for the road surface as it happened to be just before t3 or, similarly, the query that asks for the address history of our citizen according to our perception just before t3. This "historical" query, which we also encountered in our applications in practice, can now simply be expressed as follows:

Surface(X2), where X2 = $\{(I;t) \in X \mid t \prec t3\}$, a subset of X.

2. Representation in databases

We now turn our attention to *representations* of (ordered) interval sets. An (ordered) interval set might for example be represented by some table R with attributes BP and EP for begin point and end point (and TP for "time point" in the ordered case), where each individual tuple represents an (ordered) interval. The table may have other attributes as well, attributes that represent additional information about the individual intervals. As a spatial example, intervals that represent road segments may also have an attribute "road width" or an attribute "traffic intensity" (as in [Br92]). As a temporal example, intervals that represent medical treatments may also have attributes "patient number" and "treatment code" (see, e.g., [Br95], Chapter 4). As a spatio-temporal example, intervals that represent road layer segments may also have attributes "surface material" and "lay-out date" (as in [Br92]).
Of course, several other representations might be possible as well, e.g., a representation by a table with a "nested" attribute INTERVAL in stead of the attributes BP and EP. For various other examples we refer to [LM95] and [Br95].
Therefore, in general, we say for a function f and sets A and B that:

(a) A is a *representation of* B *under* f iff f is a function *from* A *onto* B,
i.e., A = dom(f) and B = rng(f);
(b) A is a *one-to-one representation of* B *under* f iff f is a *bijection* from A onto B;
(c) a is a *representative of* b *under* f iff a ∈ dom(f) and f(a) = b.

The table R1 below, with attributes BP, EP, and TP (and some other attributes), can be considered as a representation of the ordered interval set X1 from Figure 3 under the function f1 that assigns to each tuple t in R1 the ordered interval with begin point t(BP), end point t(EP), and "time point" t(TP). Formally: f1 = λt ∈ R1: ([t(BP)..t(EP)); t(TP)).
Here we use the notation "λt ∈ A: u_t" (where A represents a set and u_t is some expression in t) as an abbreviation for the function that assigns to each t ∈ A the value u_t, i.e., the function "{(t; u_t) | t ∈ A }".
Since the attribute set {BP, EP, TP} is uniquely identifying in R1, the representation is one-to-one.

BP	**EP**	**TP**	**A**	**B**	**C**
d1	d2	t3	a1	b1	c1
d2	d4	t2	a2	b2	c2
d5	d7	t2	a3	b3	c3
d9	d10	t2	a4	b4	c4
d11	d12	t3	a5	b5	c5
d1	d3	t1	a6	b6	c6
d3	d6	t1	a7	b7	c7
d8	d13	t1	a8	b8	c8

Figure 5: A table representing the ordered interval set X1 from Figure 3

We will now show in general how our notions regarding interval sets and ordered interval sets from Section 1 can be "translated" to their table representations. After our careful mathematical analysis in Section 1, this translation reduces in most cases to a straightforward substitution exercise, as we will show below.

As an intro to such a substitution exercise for interval sets, let us suppose that a table R with attributes BP and EP is a one-to-one representation of some interval set S under the function f defined by:

$f = \lambda t \in R: [t(BP)..t(EP))$.

Under this representation, the requirement in Section 1.1 that $b < e$ for an interval $[b..e)$ - in order to avoid invalid intervals - translates to the tuple constraint $t(BP) < t(EP)$ in R. (The requirement would translate to an attribute constraint in the case of a representation by our "nested" attribute INTERVAL mentioned earlier.)

Let the tuples t and t′ in R represent the intervals $[b..e)$ and $[b'..e')$, respectively. Then we can use the following substitutions in Section 1:

$b = t(BP)$, $e = t(EP)$, $b' = t'(BP)$, and $e' = t'(EP)$.

For the tuples t and t′ in R we then get that

(a) t and t′ represent *disjoint* intervals iff $t(EP) \le t'(BP)$ or $t'(EP) \le t(BP)$.

(b) t and t′ represent *intersecting* intervals iff $t'(BP) < t(EP)$ and $t(BP) < t'(EP)$.

(c) t represents an interval that *includes* t′ iff $t(BP) \le t'(BP)$ and $t'(EP) \le t(EP)$.

(d) t and t′ represent *adjacent* intervals iff $t'(BP) = t(EP)$ or $t(BP) = t'(EP)$.

(e) t and t′ represent *separate* intervals iff $t(EP) < t'(BP)$ or $t'(EP) < t(BP)$.

The translations of the following notions from Section 1.2 are perhaps less simple to understand than those from Section 1.1, but as simple to deduce:

R represents a *canonical* set of intervals iff

$\forall t \in R: \forall t' \in R:$ if $t \neq t'$ then $t(EP) < t'(BP)$ or $t'(EP) < t(BP)$.

Similarly, R represents an *intersection-free* set of intervals iff

$\forall t \in R: \forall t' \in R:$ if $t \neq t'$ then $t(EP) \le t'(BP)$ or $t'(EP) \le t(BP)$.

The sets Begset(S), Endset(S), and Bndset(S) are (represented by) the sets Begrep(R), Endrep(R), and Bndrep(R) defined as follows:

$Begrep(R) = \{ t(BP) \mid t \in R \}$

$Endrep(R) = \{ t(EP) \mid t \in R \}$

$Bndrep(R) = Begrep(R) \cup Endrep(R)$

As noted before, the table R as a representation of an interval set S may also have other (interval-related) attributes besides BP and EP, e.g., "surface material used" and "lay-out date" in case of road layer segments. However, as illustrated in Figure 2, a minimal (resp. maximal) interval might well be the intersection (resp. union) of several other intervals, so for any given attribute of R the representations of those different intervals will usually have different values for that attribute. We therefore define Minintrep(R) and Maxintrep(R), the representations of Minintset(S) and Maxintset(S), as tables over the attribute set {BP, EP} only.
So, given the representation R of the interval set S, we define Minintrep(R), the representation of Minintset(S), as the following table over the attribute set {BP, EP}:

Minintrep(R) = { {(BP;x), (EP;y)} | x ∈ Bndrep(R) and y ∈ Bndrep(R) and
x ◁ y in Bndrep(R) and ∃t ∈ R: t(BP) ≤ x and y ≤ t(EP) }

The sets Prpbegset(S), Prpendset(S), and Prpbndset(S), i.e., the sets of all *proper begin, end,* and *boundary points* in S, respectively, are (represented by) the sets Prpbegrep(R), Prpendrep(R), and Prpbndrep(R) defined as follows:

Prpbegrep(R) = { t(BP) | t ∈ R and ∀t′ ∈ R: (if t′(BP) < t(BP) then t′(EP) < t(BP)) }
Prpendrep(R) = { t(EP) | t ∈ R and ∀t′ ∈ R: (if t(EP) < t′(EP) then t(EP) < t′(BP)) }
Prpbndrep(R) = Prpbegrep(R) ∪ Prpendrep(R)

Now we are also able to define Maxintrep(R), the representation of Maxintset(S):

Maxintrep(R) = { {(BP;b), (EP;e)} | b ∈ Prpbegrep(R) and e ∈ Prpendrep(R) and
b ◁ e in Prpendrep(R) }

In order to translate the notions in Section 1.3 to their table representations, let us now suppose that a table R with attributes BP, EP, and TP is a one-to-one representation of some *ordered* interval set X under the function f defined by:

f = λt ∈ R: ([t(BP)..t(EP)); t(TP)).

Then Intset(X), the *interval set of* X, will be represented by the table R "minus" the TP-column, denoted by Intrep(R) and defined as follows:

Intrep(R) = { t ⩤ {TP} | t ∈ R },

where $t \succ\!\!\!\circ B$ denotes for any attribute set B the tuple t "minus" the B-attributes (and their values):

$$t \succ\!\!\!\circ B = \{ (a;v) \in t \mid a \notin B \}.$$

The sets Visbegset(X), Visendset(X), and Visbndset(X), i.e., the sets of all *visible begin, end,* and *boundary points* in X, respectively, are (represented by) the sets Visbegrep(R), Visendrep(R), and Visbndrep(R) defined as follows:

$$\text{Visbegrep}(R) = \{ t(BP) \mid t \in R \text{ and } \neg \exists t' \in R: (t'(BP) \le t(BP) \text{ and } t(BP) \le t'(EP) \text{ and } t(TP) \prec t'(TP)) \}$$

$$\text{Visendrep}(R) = \{ t(EP) \mid t \in R \text{ and } \neg \exists t' \in R: (t'(BP) \le t(EP) \text{ and } t(EP) \le t'(EP) \text{ and } t(TP) \prec t'(TP)) \}$$

$$\text{Visbndrep}(R) = \text{Visbegrep}(R) \cup \text{Visendrep}(R)$$

Now we turn our attention to the database representation of visible *intervals*. From Section 1.3 we recall that a visible interval is assembled from two successive visible boundary points x and y and the most recent ordered interval that covers [x..y). Given the representation R of X, such an ordered interval is represented by a tuple r in the table R (which, as we pointed out, may have other attributes as well). So, the *visible* interval is represented by r but with boundary points x and y. In our notation:

$$(r \succ\!\!\!\circ \{BP, EP\}) \cup \{(BP;x), (EP;y)\}.$$

We can also write: $r \theta \{(BP;x), (EP;y)\}$, where θ denotes *functional overriding*, see [Br95] or [Sp89]. This brings us to the following definition of the representation of the surface:

$$\underline{\text{Surfacerep}(R)} = \{ r \theta \{(BP;x), (EP;y)\} \mid x \in \text{Visbndrep}(R) \text{ and } y \in \text{Visbndrep}(R) \text{ and } r \in R \text{ and } r(BP) \le x \text{ and } y \le r(EP) \text{ and } x \lhd y \text{ in Visbndrep}(R) \text{ and } \neg \exists r' \in R: (r'(BP) \le x \text{ and } y \le r'(EP) \text{ and } r(TP) \prec r'(TP)) \}$$

When we apply this definition to our table R1 in Figure 5, which is a representation of the ordered interval set X1 from Figure 3, then we get the following result (which is in line with Figure 4 again):

BP	EP	TP	A	B	C
d1	d2	t3	a1	b1	c1
d2	d4	t2	a2	b2	c2
d4	d5	t1	a7	b7	c7
d5	d7	t2	a3	b3	c3
d8	d9	t1	a8	b8	c8
d9	d10	t2	a4	b4	c4
d10	d11	t1	a8	b8	c8
d11	d12	t3	a5	b5	c5
d12	d13	t1	a8	b8	c8

Figure 6: The representation of the surface of X1

Our business rule BR1 from Section 1.3 applied to R translates to the following table constraint:

$\forall r \in R: \forall r' \in R:$ if $r(TP) = r'(TP)$ and $r(BP) < r'(EP)$ and $r'(BP) < r(EP)$
then $r(BP) = r'(BP)$ and $r(EP) = r'(EP)$

Finally we can translate the "historical surface" query at the end of Section 1.3 in a simple way too:

Surfacerep(R2), where $R2 = \{ r \in R \mid r(TP) \prec t3 \}$, a subset of R.

3. Realization in SQL

In order to be able to implement our ideas in practice, we will now show how our database representations of interval sets and ordered interval sets from Section 2 can be translated into SQL. Here we will apply the general translation rules from [Br95], Chapter 9. As an additional policy, we could turn the definitions of the notions in Section 2 into view definitions (or rather, view templates), thus creating a view system in a structured way.

To start with some simple illustrations, we define three views called BEGVW-R, ENDVW-R, and BNDVW-R (each with one attribute named PNT) of all begin points, end points, and boundary points of a table R:

```
CREATE VIEW  BEGVW-R(PNT)  AS  (SELECT  DISTINCT  BP  FROM  R)
CREATE VIEW  ENDVW-R(PNT)  AS  (SELECT  DISTINCT  EP  FROM  R)
CREATE VIEW  BNDVW-R(PNT)  AS  (SELECT  *  FROM  BEGVW-R)  UNION
                               (SELECT  *  FROM  ENDVW-R)
```

We note that, unlike the SELECT, the result of the UNION never contains duplicate elements in SQL.

Most translation rules are fairly straightforward. Perhaps less trivial are the translation rules for the quantifiers ∃ and ∀. We will repeat them here.

A Boolean expression of the form $\exists t \in R: \varphi(t)$ can for instance be translated to

```
EXISTS( SELECT  t.*  FROM  R  t  WHERE  φ′(t) )
```

where t is used as an alias and $\varphi'(t)$ is the translation of $\varphi(t)$ into SQL.

Similarly, a Boolean expression of the form $\forall t \in R: \varphi(t)$ can be translated to

```
NOT EXISTS( SELECT  t.*  FROM  R  t  WHERE NOT (φ′(t)) )        or for instance to:
0  =  SELECT  COUNT(t.*)  FROM  R  t  WHERE NOT (φ′(t))
```

As an example of a "∀-translation" we will translate Prpbegset(S), the set of all *proper begin points* in S. In Section 2 we derived that Prpbegset(S) is represented by the set

$\{ t(BP) \mid t \in R \text{ and } \forall t' \in R: (\text{if } t'(BP) < t(BP) \text{ then } t'(EP) < t(BP)) \}$.

According to our rules (and replacing t′ by tt, in behalf of SQL), this can simply be translated to a view:

```
CREATE VIEW  PRPBEGVW-R(PNT)  AS
SELECT  DISTINCT  t.BP  FROM  R  t
WHERE  NOT  EXISTS(  SELECT  tt.*  FROM  R  tt
                          WHERE  NOT (tt.BP >= t.BP OR tt.EP < t.BP) )
```

Note that the inner WHERE-clause can be rewritten to: tt.BP < t.BP AND tt.EP >= t.BP.
A view called PRPENDVW-R could be created in a similar way.
Based on these two views, a view called PRPBNDVW-R could then be created as follows:

```
CREATE VIEW  PRPBNDVW-R(PNT)  AS
(SELECT  *  FROM  PRPBEGVW-R)  UNION  (SELECT  *  FROM  PRPENDVW-R)
```

As an example of an "∃-translation", we will translate Visbegset(X), the set of all *visible begin points* in X. In Section 2 we derived that Visbegset(X) is represented by the set

$$\{\ t(BP)\ |\ t \in R \text{ and } \neg\, \exists t' \in R\colon (t'(BP) \le t(BP) \text{ and } t(BP) \le t'(EP) \text{ and } t(TP) \prec t'(TP))\ \}$$

According to our rules (and again replacing t′ by tt), this can simply be translated to a view as well:

```
CREATE VIEW  VISBEGVW-R(PNT)  AS
SELECT  DISTINCT  t.BP  FROM  R  t
WHERE  NOT  EXISTS(  SELECT  tt.*  FROM  R  tt
                          WHERE  tt.BP =< t.BP AND t.BP =< tt.EP AND t.TP ≺ tt.TP )
```

A view called VISENDVW-R could be created in a similar way.
Based on these two views, a view called VISBNDVW-R could then be created as follows:

```
CREATE VIEW  VISBNDVW-R(PNT)  AS
(SELECT  *  FROM  VISBEGVW-R)  UNION  (SELECT  *  FROM  VISENDVW-R)
```

In order to translate Minintset, Maxintset, and Surface, we recall that $x \lhd y$ in V stands for

$$x < y \text{ and } \neg\, \exists z \in V\colon (x < z \text{ and } z < y).$$

According to our translation rules this Boolean expression translates to something of the form

```
x < y AND NOT EXISTS( SELECT  z.*  FROM  V  z
                          WHERE  x < z  AND  z < y )
```

When we would substitute this in the definitions of Minintrep(R), Maxintrep(R), and Surfacerep(R) in Section 2, we get:

```
CREATE VIEW  MININTVW-R(BP, EP)  AS
SELECT  b.PNT  AS  BP,
        e.PNT   AS  EP
FROM    BNDVW-R  b,  BNDVW-R  e
WHERE  b.PNT < e.PNT
   AND  NOT  EXISTS(  SELECT  z.*  FROM  BNDVW-R  z
                      WHERE  b.PNT < z.PNT  AND  z.PNT < e.PNT )
   AND  EXISTS(  SELECT  t.*  FROM  R  t
                 WHERE  t.BP =< b.PNT  AND  e.PNT =< t.EP )

CREATE VIEW  MAXINTVW-R(BP, EP)  AS
SELECT  b.PNT  AS  BP,
        e.PNT   AS  EP
FROM    PRPBEGVW-R  b,  PRPENDVW-R  e
WHERE  b.PNT < e.PNT
   AND  NOT  EXISTS(  SELECT  z.*  FROM  PRPENDVW-R  z
                      WHERE  b.PNT < z.PNT  AND  z.PNT < e.PNT )

CREATE VIEW  SURFACEVW-R(Ω, BP, EP)  AS
SELECT  DISTINCT  Ω,
        b.PNT  AS  BP,
        e.PNT   AS  EP
FROM    VISBNDVW-R  b,  VISBNDVW-R  e,  R  t
WHERE  t.BP =< b.PNT  AND  e.PNT =< t.EP  AND  b.PNT < e.PNT
   AND  NOT  EXISTS(  SELECT  z.*  FROM  VISBNDVW-R  z
                      WHERE  b.PNT < z.PNT  AND  z.PNT < e.PNT )
   AND  NOT  EXISTS(  SELECT  tt.*  FROM  R  tt
                      WHERE  tt.BP =< b.PNT AND e.PNT =< tt.EP AND t.TP ≺ tt.TP
                )
```

where Ω denotes the other attribute names of the R-tuple t (separated by commas).

According to our translation rules from [Br95], the special business rule BR1 from the previous section can be translated to a so-called *assertion definition* in SQL (i.e., at data definition time). After applying our ∀-translation rule twice as well as some rewriting rules from logic we get:

```
CREATE ASSERTION  BR1
CHECK( NOT EXISTS( SELECT  t.*  FROM  R  t
                        WHERE EXISTS( SELECT  tt.*  FROM  R  tt
                                          WHERE t.TP = tt.TP AND t.BP < tt.EP
                                              AND tt.BP < t.EP
                                              AND NOT (t.BP = tt.BP AND t.EP = tt.EP) )
                                          ) )
```

Now we have a look at our “historical surface” query at the end of the previous section, which asked for the surface of a subset R2 of the table R. The subset R2 can be translated to the following view:

```
CREATE VIEW  R2  AS
SELECT  t.*  FROM  R t  WHERE  t.TP ≺ t3
```

Now we want to apply the surface-definition to R2 in stead of R. This can be done by textual substitution in the text of the surface-definition and of all its underlying auxiliary definitions (VISBNDVW and, subsequently, VISBEGVW and VISENDVW):

```
CREATE VIEW  SURFACEVW-R2(Ω, BP, EP)  AS
SELECT  DISTINCT  Ω,
          b.PNT  AS  BP,
          e.PNT   AS  EP
FROM    VISBNDVW-R2  b,  VISBNDVW-R2  e,  R2  t
WHERE  t.BP =< b.PNT  AND  e.PNT =< t.EP  AND  b.PNT < e.PNT
    AND  NOT  EXISTS(  SELECT  z.*  FROM  VISBNDVW-R2  z
                            WHERE  b.PNT < z.PNT  AND  z.PNT < e.PNT )
    AND  NOT  EXISTS(  SELECT  tt.*  FROM  R2  tt
```

```
                    WHERE  tt.BP =< b.PNT AND e.PNT =< tt.EP AND t.TP ≺ tt.TP
          )
```

This analysis shows that it would be useful, for a developer for instance, to have the table concerned as a parameter of these operations. These operations could then simply be called with that table as a parameter (in stead of defined by textual substitution in the text of the original definition and of all its underlying auxiliary definitions).

To end this section on realization in SQL with some suggestions for improvement, we note that the description of a set such as Minintrep(R) is of the form

$\{ u \mid x \in B \text{ and } y \in B \text{ and } x \lhd y \text{ in } B \text{ and } \varphi \}$

where u and the Boolean φ are typically expressions in x and y. According to our translation rules this translates to an SQL-expression of the form

```
SELECT  [DISTINCT] u′
FROM    V x,  V y
WHERE  x.PNT < y.PNT  AND
          NOT   EXISTS( SELECT  z.*  FROM  V z
                              WHERE  x.PNT < z.PNT  AND  z.PNT < y.PNT )
   AND  φ′
```

where V is a table or view (representing B) with an attribute PNT.

If n is the number of rows of V then the SQL-expression above is of the order n^3 (due to the join and the subselect over V). However, when we can make use of *cursors* in SQL (see, e.g., [CO92]) then we can reduce the translation from a cubic order to a lineair order! We will sketch the idea in terms of the following pseudo-code:

```
DECLARE  C  CURSOR
FOR      <SQL-statement describing V>
         ORDER BY  PNT;

OPEN   C;
FETCH  C  INTO  x;
IF  found(C)  THEN  FETCH  C  INTO  y;
WHILE  found(C)
    DO  BEGIN  /* A */
        IF  φ′  THEN  return(u′);
        x := y;
        FETCH  C  INTO  y
        END;
CLOSE  C
```

Explanation:
First, the cursor C is declared, i.e., defined as a certain table expression describing V, ordered by PNT (in ascending order).
Then the cursor C is opened, i.e., its contents is computed.
The FETCH statements assign the next tuple (if any) in the result set to a variable.
The Boolean expression found(C) returns TRUE iff the latest FETCH did return a tuple.
At point /* A */ the following invariant holds: $x \in B$ and $y \in B$ and $x \lhd y$ in B.
If φ′ holds then u′ is added to the final outcome.
Then x and y are moved on one step in the (ordered) result set.
Finally the cursor C is closed again.
The statement return(u′) could be replaced by, e.g., an insert(-statement) into some other result table.

4. Comparison with related work

The paper [LM95] presents a checklist of functional requirements for interval databases (in particular Valid Time models) and evaluates several approaches against this checklist. We will take this checklist as our starting point for placing our approach in context. By ‘our approach’

we mean here the contents of our paper in combination with the underlying database approach as presented in [Br95].

As a general remark, our approach can be characterized as a non-1NF approach (i.e., not restricted to first normal form, see the explanation on page 83 of [Br95]) that also includes generic intervals (see this paper). Therefore our approach falls into the most general category that [LM95] distinguishes (called I-NESTED).

We will now check our model against the individual functional requirements in [LM95], which will be repeated below. For the ease of reference, we adopt the same requirement codes (G1, G2, T1, etc.).

As already noted in [LM95], the general property (G1), which is inherited from [TL82], is too subjective and is therefore assumed to hold for all models. Apart from this argument, we already discussed the abstract and (relatively) simple character of our model, and we also showed that our operations are closed (in the (a)-clauses of our Propositions). So,

(G1) our model is abstract and simple and its operations are closed.

As explained earlier (e.g., in Section 1.3),

(G2) it is possible to record data valid in the past, present, and future in a table in our model.

We already discussed in the beginning of Section 1.1 that, although not necessary,

(T1) it is possible to consider valid time as discrete and totally ordered in our model.

We also discussed in that section that

(T2) our model can support various granularities for valid time.

Although it is not necessary,

(T3) the use of valid time could be restricted exclusively to either time points or time intervals in our model.

In a discrete total order with a notion of successor, we can rewrite the other three types of intervals to left-closed, right-open intervals as follows:

$[x..y] = [x..succ(y)), \quad (x..y) = [succ(x)..y), \text{ and } \quad (x..y] = [succ(x)..succ(y))$.

More generally, if there is also a notion of predecessor, we (or a user-friendly interface) can rewrite any of the four interval types to any other one by the following rules:

from	(x..	to	[succ(x)..
from	..y)	to	..pred(y)]
from	[x..	to	(pred(x)..
from	..y]	to	..succ(y))

Consequently, for reasons of user-friendliness (mentioned as an argument in [LM95]),

(T5) in a discrete total order with a notion of successor and predecessor, all four types of intervals could directly be supported with our model.

Since our database model incorporates Codd's classical relational model,

(D1) every data type valid in Codd's model is also valid in our model,

(V1) the semantics, constraints, and functions of Codd's model can be enhanced in our model,

(O1) the operations Union, Difference, Projection, and Cartesian Product remain the same in our model, and

(S1) every relation in Codd's model is also valid in our model.

It will be clear that as far as the data structures are concerned,

(S2) tables with more than one valid time attribute (of possibly different time granularities) of either point or interval type are possible in our model.

In Section 1.1 we showed how interval predicates (such as adjacency, separateness, inclusion, and others) can be added to our lingo, so

(O2) in our model the Selection operation can be generalized by the inclusion of interval predicates.

In a finite total order with a notion of successor, every point x can be transformed into the one-point interval [x..succ(x)) and an interval set S can be transformed into $\bigcup S$, the point set of S. Hence,

(O4) in a finite total order, operations in our model enable transformations between points and intervals.

We only treat the "union" (P-Union in [LM95]) and "intersection" (P-Project in [LM95]), for one-dimensional interval sets (although we additionally treat the notions of "visibility" and "surface" for ordered interval sets); therefore we only partly satisfy the requirement that

(O3) operations like P-Union, P-Project, and P-Difference have to be defined, for one-dimensional as well as multi-dimensional intervals.

Like the other models evaluated in [LM95], our model does not satisfy the requirement (T4). Although it follows from our discussion in Section 2 (on avoiding invalid intervals and on using an attribute INTERVAL) as well as from the properties (O2) and (S2) discussed above that our model does not have all the drawbacks mentioned in [LM95], it does not really satisfy the requirement that

(T4) a valid time interval should be supported as a primitive data type.

In [LM95], Lorentzos and Manolopoulos summarize their evaluation of several approaches against their checklist presented in that paper. Figure 7 below results from adding our evaluation of our approach to their summary table. The possible answers (and their meaning) are as follows:
Y (Yes), **P** (Partly), **?** (not specified in the literature), and **N** (No).

Model	**G1**	**G2**	**T1**	**T2**	**T3**	**T4**	**T5**	**D1**	**S1**	**S2**	**O1**	**O2**	**O3**	**O4**	**V1**
Ben-Zvi	Y	Y	Y	Y	N	N	N	Y	Y	N	?	P	?	N	P
Sadeghi	Y	?	Y	Y	N	N	N	Y	Y	N	P	Y	?	N	P
Jones	Y	Y	Y	Y	N	N	N	Y	Y	N	?	Y	?	N	P
Snodgrass	Y	Y	Y	Y	Y	N	N	Y	Y	N	P	Y	?	N	P
Navathe	Y	Y	Y	Y	N	N	N	Y	Y	N	P	Y	Y	P	P
Sarda	Y	Y	Y	Y	Y	N	N	Y	Y	N	Y	Y	Y	P	P
Lorentzos	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	P
De Brock	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	P	Y	Y

Figure 7: Evaluation of several models

The table below summarizes the number of answers **Y** (Yes), **P** (Partly), **?** (not specified in the literature), and **N** (No) per approach.

Model	**Y**	**P**	**?**	**N**
Ben-Zvi	6	2	2	5
Sadeghi	6	2	2	5
Jones	7	1	2	5
Snodgrass	8	2	1	4
Navathe	8	3	-	4
Sarda	10	2	-	3
Lorentzos	12	1	-	2
De Brock	13	1	-	1

Figure 8: Summary of the answers in Figure 7

When we compare our approach with Lorentzos' in [LP94] and [LP95] for instance, we see that [LP94] treats the "union" and "difference" of multi-dimensional interval sets whereas we treat the "union" and "intersection" of one-dimensional interval sets as well as "visibility" and the "surface" of ordered interval sets. Additional research could be done to extend our approach to multi-dimensional interval sets.
We do not require that a lineair domain is finite or even discrete, which makes our approach more general than many other approaches (see, e.g., [Ga88], [MS91], [CC93], [TC93], [LP95], [EG98]).

Conclusions

We separated the concern of treating intervals and interval operations in their clean mathematical context from their representations in a (relational) data model, and their translations into SQL. This approach simplifies the overall complexity of the proper treatment of intervals considerably, and it avoids the need to extend the relational model or SQL (like some other approaches do). Once we have our mathematical definitions right and their properties established, it is relatively straightforward to translate them into the relational model and, subsequently, into SQL. So, the determination of the right set of definitions and their relevant properties is one of the contributions of our paper. In summary, the idea is to break down the translation consequently into the following distinct steps (where "→" means "are represented by"):

real world objects → mathematical intervals → database representations → SQL realizations

Perhaps that constitutes the most important contribution of this paper: it introduces a much simpler theory for interval databases. As another illustration, [LP94], page 169, states that the specific order of their folding is part of the semantics. However, we would say that it only concerns *different representations* of (semantically) *the same information*: in their example, it turns out that the employee has to work the same hours in their different SHIFT-solutions (of course); in other words, these solutions are point-wise equivalent.
Another advantage of our approach is that there is no need for so-called *basic* or *elementary* intervals, i.e., intervals consisting of only one "point" (see [LP94 for instance). In our road layer segments application this would have forced us to choose some suitable basic unit

length, e.g., 1 meter, and this would subsequently lead to many thousands of unit segments … Fortunately, our approach leads to more practical (and more efficient) solutions.

In conclusion, we succeeded in our goal to give a clear, declarative, and relatively simple treatment of various practically useful operations on (ordered) interval sets and to provide straightforward realizations in standard SQL. We did so by making a clear distinction between abstract (mathematical) intervals and their (database) representations, and by providing and applying general rules for translating our database constructs directly into standard SQL. We also introduced some new interval-related notions (ordered interval set, visibility, and surface of an ordered interval set), which are important in practical applications. We adequately treated their properties and functionality.

References

[Ar86] G. Ariav: A temporally oriented datamodel.
ACM Transactions on Database Systems, vol. 11, 1986, pp. 499-527

[AS93] K.K. Al-Taha, R.T. Snodgrass, and M.D. Soo:
Bibliography on spatiotemporal databases.
ACM SIGMOD Record, vol. 22, 1993, pp. 59-67

[Be82] J. Ben-Zvi: The time relational model.
Ph.D. dissertation, Dep. of Computer Science, UCLA, Los Angeles, 1982

[Br92] E.O. de Brock: System Specification WIS (Road Information System).
Specification report for the Polder of North- and South-Beveland, 1992 (In Dutch)

[Br95] E.O. de Brock: Foundations of Semantic Databases.
Prentice Hall International Series in Computer Science, Hemel Hempstead, 1995

[CC93] J. Clifford and A. Croker: The historical relational data model (HRDM) revisited.
In [TC93], pp. 6-27

[CO92] S.J. Cannan and G.A.M. Otten: SQL, the standard handbook.
McGraw-Hill, London, 1992

[DH80] S.M. Deen and P. Hammersley (Eds.): Proc. of the Int. Conf. on Data Bases.
British Computer Society, 1980

[EG98] O. Etzion, A. Gal, and A. Segev:
Extended update functionality in temporal databases. In [EJ98], pp. 56-95

[EJ98] O. Etzion, S. Jajodia, and S. Sripada (Eds.):
Temporal databases: research and practice.
Springer, Berlin, LNCS 1399, 1998

[Ga88] S.K. Gadia:
A homogeneous relational model and query languages for temporal databases.
ACM Transactions on Database Systems, vol. 13, 1988, pp. 418-448

[JM80] S. Jones and P.S. Mason: Handling the time dimension in a database.
In [DH80], pp. 65-83

[KL00] G.M. Kuper, L. Libkin, and J. Paredaens (Eds.): Constraint Databases.
Springer Verlag, Berlin, 2000

[LJ88] N.A. Lorentzos and R.G. Johnson:
An extension of the relational model to support generic intervals.
In [SC88], pp. 528-542

[LJ88b] N.A. Lorentzos and R.G. Johnson:
TRA, a model for a temporal relational algebra. In [RB88], pp. 203-215

[LM95] N.A. Lorentzos and Y. Manolopoulos: Functional requirements for historical and interval extensions to the relational model.
Data & Knowledge Engineering, vol. 17, 1995, pp. 59-86

[LP94] N.A. Lorentzos, A. Poulovassilis, and C. Small:
Implementation of update operations for interval relations.
The Computer Journal, vol. 37, 1994, pp. 164-176

[LP95] N.A. Lorentzos, A. Poulovassilis, and C. Small:
Manipulation operations for an interval-extended relational model.
Data & Knowledge Engineering, vol. 17, 1995, pp. 1-29

[MS91] L.E. McKenzie and R.T. Snodgrass:
Evaluation of relational algebras incorporating the time dimension in databases.
ACM Computing Surveys, vol. 23, 1991, pp. 501-543

[NA93] S.B. Navathe and R. Ahmed:
Temporal extensions to the relational model and SQL.
In [TC93], pp. 92-109

[RB88] C. Rolland, F. Bodart, and M. Leonard (Eds.):
Temporal aspects in information systems. North Holland, 1988

[Si87] R. Sadeghi: A database query language for operations on historical data.
Ph.D. dissertation, Dundee College of Technology, 1987

[Sa90] N.L. Sarda: Extensions to SQL for historical databases.

IEEE Trans. on Knowledge and Data Engineering, 1990, pp. 220-230

[Sa90b] N.L. Sarda: Algebra and query language for a historical data model.
The Computer Journal, vol. 33, 1990, pp. 11-18

[SC88] J.W. Schmidt, S. Ceri, and M. Missikoff (Eds.): Advances in database technology.
Proc. EDBT'88, Springer, Berlin, LNCS 303, 1988

[SB98] R.T. Snodgrass, M.H. Böhlen, C.S. Jensen, and A. Steiner:
Transitioning temporal support in TSQL2 to SQL3. In [EJ98], pp. 150-194

[Sn87] R. Snodgrass: The temporal query language TQuel.
ACM Transactions on Database Systems, vol. 12, 1987, pp. 247-298

[Sn95] R.T. Snodgrass (Ed.): The TSQL2 Temporal Query Language.
Kluwer Academic Publishers, Dordrecht, 1995

[Sp89] J.M. Spivey: The Z Notation: A Reference Manual.
Prentice Hall, Hemel Hempstead, 1989

[TC93] A.U. Tansel, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass (Eds.):
Temporal Databases: Theory, Design, and Implementation.
Benjamin/Cummings, 1993

[TL82] D.C. Tsichritzis and F.H. Lochovsky: Data Models.
Prentice Hall, New Jersey, 1982

[To00] D. Toman:
Point-based temporal extensions of SQL and their efficient implementation.
In [KL00], pp. 391-399

[WJ98] Y. Wu, S. Jajodia, and X.S. Wang: Temporal database bibliography update.
In [EJ98], pp. 338-366